

# Introduction to cryptology (GBIN8U16)



## Hash functions

Pierre Karpman

`pierre.karpman@univ-grenoble-alpes.fr`

`https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html`

2020-03

# Hash functions as a figure

---

≈ on the board

# First definition

## Hash function

A hash function is a mapping  $\mathcal{H} : \mathcal{M} \rightarrow \mathcal{D}$

So it really is just a function...

Usually:

- ▶  $\mathcal{M} = \bigcup_{\ell < N} \{0, 1\}^\ell$ ,  $\mathcal{D} = \{0, 1\}^n$ ,  $N \gg n$
- ▶  $N$  is typically  $\geq 2^{64}$ ,  $n \in \{128, 160, 224, 256, 384, 512\}$

Also popular now: extendable-output functions (XOFs):  $\mathcal{D} = \bigcup_{\ell < N'} \{0, 1\}^\ell$

- ▶ Hash functions are *keyless*
- ▶ So, how do you tell if one's **good**?

## Random oracle

A function  $\mathcal{H} : \mathcal{M} \rightarrow \mathcal{D}$  s.t.  $\forall x \in \mathcal{M}, \mathcal{H}(x) \stackrel{\$}{\leftarrow} \mathcal{D}$

- ▶ “The best we can ever get”
- ▶ Sometimes useful in proofs (“Random oracle model”, or ROM)
- ▶ Not possible to have one (except for small (co-)domains assuming a TRNG)
- ▶ But we can get *approximations* (e.g. SHA-3)
- ▶ Equivalent to the Ideal Cipher Model (Coron et al., 2008; + later patches)

# Main security properties

---

What is hard for a RO should be hard for any HF

⇒

- 1 **First preimage**: given  $t$ , find  $m$  s.t.  $\mathcal{H}(m) = t$
- 2 **Second preimage**: given  $m$ , find  $m' \neq m$  s.t.  $\mathcal{H}(m) = \mathcal{H}(m')$
- 3 **Collision**: find  $(m, m' \neq m)$  s.t.  $\mathcal{H}(m) = \mathcal{H}(m')$

Generic complexity:

1), 2):  $\Theta(2^n)$ ;

3):  $\Theta(2^{n/2}) \leftarrow$  “Birthday paradox”

# Why do we care? Applications!

---

Hash functions are useful for:

- ▶ Hash-and-sign (RSA signatures, (EC)DSA, ...)
- ▶ Message-authentication codes (HMAC, ...)
- ▶ Password hashing (with a grain of salt)
- ▶ Hash-based signatures (inefficient but PQ)
- ▶ As “RO instantiations” (OAEP, ...)
- ▶ As one-way functions (OWF)

# So, how do you build hash functions?

---

- ▶ Objective #1: be secure
- ▶ Objective #2: be efficient
  - ▶ At most a few dozen cycles/byte!
  - ▶  $\Rightarrow$  work with limited amount of memory

So...

- ▶ (#2) Build  $\mathcal{H}$  from a *small component*
- ▶ (#1) Prove that this is okay

# What kind of small component?

---

## Compression function

A compression function is a mapping  $f : \{0,1\}^n \times \{0,1\}^b \rightarrow \{0,1\}^n$

- ▶ A family of functions from  $n$  to  $n$  bits
- ▶ Not unlike a block cipher, only not invertible

## Permutation

A permutation is an invertible mapping  $p : \{0,1\}^n \rightarrow \{0,1\}^n$

Yes, very simple

- ▶ Like a block cipher with a fixed key, e.g.  $p = \mathcal{E}(0, \cdot)$



## From small to big (compression function case)

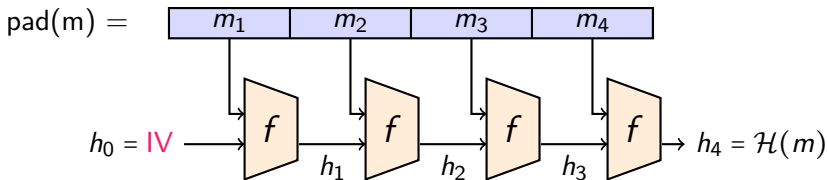
---

Assume a good  $f$

- ▶ Main problem: fixed-size domain  $\{0, 1\}^n \times \{0, 1\}^b$
- ▶ Objective: *domain extension* to  $\bigcup_{\ell < N} \{0, 1\}^\ell$
- ▶ (Not unlike using a mode of operation with a BC)

The classical answer: the Merkle-Damgård construction (1989)

## MD: with a picture



That is:  $\mathcal{H}(m_1 \| m_2 \| m_3 \| \dots) = f(\dots f(f(f(\text{IV}, m_1), m_2), m_3), \dots)$

$\text{pad}(m) \approx m \| 1000 \dots 00 \langle \text{length of } m \rangle$

# MD: does it work?

---

Efficiency?

- ▶ Only sequential calls to  $f$
- ▶  $\Rightarrow$  fine

Security?

- ▶ Still to be shown
- ▶ Objective: *reduce* security of  $\mathcal{H}$  to that of  $f$ 
  - ▶ “If  $f$  is good, then  $\mathcal{H}$  is good”
- ▶ True for collision and first preimage, **false** for second preimage

# MD (partial) security proof

Method: simple contrapositive argument

- ▶ Attack  $\{1^{\text{st}}\text{preim.}, \text{coll.}\}$  on  $\mathcal{H} \Rightarrow$  attack  $\{1^{\text{st}}\text{preim.}, \text{coll.}\}$  on  $f$

## First preimage case

If  $\mathcal{H}(m_1 \| m_2 \| \dots \| m_\ell) = t$ , then  $f(\mathcal{H}(m_1 \| m_2 \| \dots \| m_{\ell-1}), m_\ell) = t$

## Collision case (sketch)

If  $\mathcal{H}(m_1 \| m_2 \| \dots \| m_\ell) = \mathcal{H}(m'_1 \| m'_2 \| \dots \| m'_\ell)$ , show that  $\exists i$  s.t.  
 $(h_i := \mathcal{H}(m_1 \| m_2 \| \dots \| m_{i-1}), m_i) \neq (h'_i := \mathcal{H}(m'_1 \| m'_2 \| \dots \| m'_{i-1}), m'_i)$   
and  $f(h_i, m_i) = f(h'_i, m'_i)$

- ▶ Proper message padding useful to make it work!

## What about 2<sup>nd</sup> preimages??

---

No proof (with optimal resistance), can't have one:

- ▶ Generic attack on messages of  $2^k$  blocks for a cost  $\approx k2^{n/2+1} + 2^{n-k+1}$  (Kelsey and Schneier, 2005)
- ▶ Idea: exploit internal collisions in the  $h_i$ s

This is not nice, but:

- ▶ Requires (very) long messages to gain something
- ▶ At least as expensive as collision search
  - ▶ Always going to be the case, as preimage  $\Rightarrow$  collision
- ▶ If  $n$  is chosen s.t. generic collisions are out of reach, we're somewhat fine

$\Rightarrow$  Didn't make people give up MD hash functions (MD5, SHA-1, SHA-2 family)

## Is that unavoidable?

---

No! Simple patch: Chop-MD/Wide-pipe MD (Coron et al., 2005) and (Lucks, 2005)

- ▶ Build  $\mathcal{H}$  from  $f : \{0, 1\}^{2n} \times \{0, 1\}^b \rightarrow \{0, 1\}^{2n}$ , truncate output to  $n$  bits (say)
- ▶ Collision in the output  $\nRightarrow$  collision in the internal state
- ▶ Very strong provable guarantees (Coron et al.)
  - ▶ Secure domain extender for fixed-size RO
- ▶ Concrete instantiations: SHA-512/224, SHA-512/256 (2015)

## But careful with models!

---

- ▶ Coron et al. show very strong provable guarantees for Chop-MD
  - ▶ Secure domain extender for fixed-size RO
- ▶ But this in fact doesn't guarantee weaker ones, such as preservation of collision-resistance (Bellare & Ristenpart, 2006)!
  - ▶ One can do “stupid things” with a non-ideal compression function
  - ▶  $\leadsto$  Chop-MD with a (real) CR c.f. is not (necessarily) CR!
  - ▶ (In essence, one needs strengthening in the padding)

# Practical impact of the MD proof

---

- ▶ If one can't attack the  $f$  used in  $\mathcal{H}$ , all is well
- ▶ Else, ...???
- ▶  $\Rightarrow$  Attacking  $f$  is a meaningful goal for cryptographers ( $\approx$  (semi-)freestart attacks)
- ▶ Ideally, *never* use a  $\mathcal{H}$  with broken  $f$



# The MD5 failure

---

- ▶ MD5: designed by Rivest (1992)
- ▶ 1993: very efficient collision attack on the compression function (den Boer and Bosselaers); mean time of 4 minutes on a 33 MHz 80386
- ▶ MD5 still massively used...
- ▶ 2005: very efficient collision attack on the *hash function* (Wang and Yu)
- ▶ Still used...
- ▶ 2007: practically threatening collisions (Stevens et al.)
- ▶ Still used...
- ▶ 2009: even worse practical collision attacks (Stevens et al.)
- ▶ Hmm, maybe we should move on?

# Was this avoidable?

---

Yes!

- ▶ Early signs of weaknesses  $\Rightarrow$  move to alternatives ASAP!
- ▶ What were they (among others)?
  - ▶ 1992: **RIPEMD** (RIPE); practically broken (collisions) 2005 (Wang et al.)
  - ▶ 1993: **SHA-0** (NSA); broken (collisions) 1998 (Chabaud and Joux); practically broken 2005 (Biham et al.)
  - ▶ 1995: **SHA-1** (NSA); broken (collisions) 2005 (Wang et al.); practically broken 2017 (Stevens et al. (and me!))
  - ▶ 1996: **RIPEMD-128** (Dobbertin et al.); broken (collisions) 2013 (Landelle and Peyrin)
  - ▶ 1996: **RIPEMD-160** (Dobbertin et al.); unbroken so far
  - ▶ 2001: **SHA-2** (NSA); unbroken so far

## Lesson to learn?

---

- ▶ Don't use broken algorithms
- ▶ Also, broken crypto is not “cool”

Perfect bad example: Git

- ▶ Don't use SHA-1 in 2005!
- ▶ Don't hide needed security properties!

Also:

- ▶ Don't use SHA-1, even if you just care about preimage attacks

## Back to design: how to do $f$ ?

---

- 1 Start like a block cipher
- 2 Add feedforward to prevent invertibility

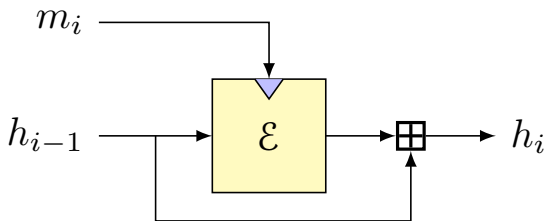
Examples:

“Davies-Meyer”:  $f(h, m) = \mathcal{E}_m(h) \boxplus h$

“Matyas-Meyer-Oseas”:  $f(h, m) = \mathcal{E}_h(m) \boxplus m$

- ▶ Systematic analysis by Preneel, Govaerts and Vandewalle (1993). “PGV” constructions
- ▶ Then rigorous proofs **in the ideal cipher model** (Black et al., 2002), (Black et al., 2010)

Picture:



Used in MD4/5 SHA-0/1/2, etc.

PGV constructions are proved secure in the *ideal cipher model*,  
**BUT**

- ▶ Real ciphers are not ideal!
- ▶ Real ciphers *don't have to be ideal* to be okay ciphers (e.g. “good” PRPs)
  - ▶ IDEA (Lai and Massey, 1991): weak key classes (Daemen et al., 1993)
  - ▶ TEA (Wheeler and Needham, 1994): equivalent keys (Kelsey et al., 1996)

What can go wrong?

## Bad case of crypto design

---

Microsoft needed a hash function for ROM integrity check of the XBOX

- ▶ Used TEA in DM mode (Steil, 2005)
- ▶ Because of an earlier break of their RC4-CBC-MAC scheme (ibid.)
- ▶ Terrible idea, because of existence of equivalent keys!
- ▶  $\text{TEA}(k, m) = \text{TEA}(\hat{k}, m) \Rightarrow \text{DM-TEA}(h, k) = \text{DM-TEA}(h, \hat{k}) \Rightarrow$  easy collisions!
- ▶ Got hacked...
- ▶ IDEA for a hash function: also bad (Wei et al., 2012)

**Never design your own crypto!**

## It's not all that bad, tho

---

- ▶ AES in a PGV construction so far unbroken (see e.g. Sasaki (2011))
  - ▶ But small parameters ?
- ▶ Ditto, SHA-256 as a block cipher: “SHACAL-2” (Handschuh and Naccache, 2001)
  - ▶ Enormous keys, 512 bit!





## And now for something different

---

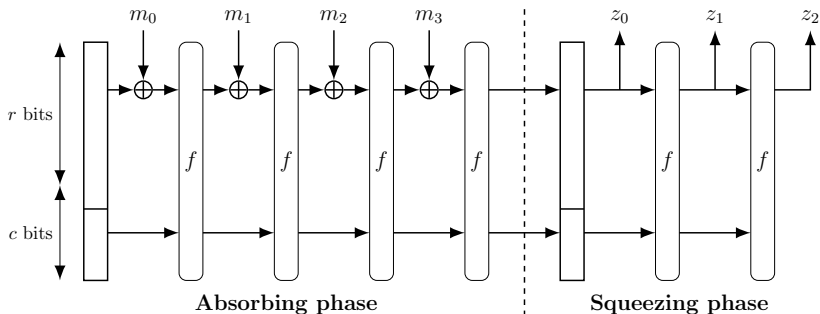
If you need a hash function today  $\Rightarrow$  SHA-3 (initially Keccak, (Bertoni et al., 2008))

- ▶ Winner of an academic competition run by NIST (2008–2012)
- ▶ Sponge construction (not Merkle-Damgård)
- ▶ Based on a permutation (not a compression function)
- ▶ Permutation is an SPN (not a Feistel, not ARX)

Sponge:

- 1 Compute  $i := p(p(\dots p(m_1 || 0^c) \oplus m_2 || 0^c) \dots)$
- 2 Output  $\mathcal{H}(m) := [i]_r || [p(i)]_r || \dots || [p^n(i)]_r$

# Picture of a sponge



<https://www.iacr.org/authors/tikz/>

## Sponge nice features

---

- ▶ Indifferentiable from a RO (same, as Wide-pipe MD) (Bertoni et al., 2008)
- ▶ Quite flexible
  - ▶ For fixed permutation size: speed/security tradeoff
- ▶ Natively a XOF
- ▶ Can be extended to do (authenticated) encryption
- ▶ Simpler to design a permutation; less of a waste?
- ▶ Close structure: JH construction, another SHA-3 competitor (Wu, 2008)

# Conclusion

---

- ▶ Don't design crypto yourself!
  - ▶ There is no generic way to design a hash function
  - ▶ Every small detail counts (recall e.g. TEA)
- ▶ Use SHA-3 (SHA-2 still okay)
- ▶ NEVER USE MD5/SHA-1
  - ▶ Even if you only care about preimage attacks